DESCRIPTION

Compiling Apparatus and Compiling Method

Technical Field

The present invention relates to a compiling apparatus for generating a program that can be executed, from a program written in a computer programming language by a computer of sequential system, parallel system or VLIW (Very Long Instruction Word) system.

Background Art

A compiling apparatus is used to generate from a program that can be executed in a particular electronic computer (hereinafter called executable program, from a source program written in a programming language, or to generate an optimal executable program from a source program.

A source program written by programmers in a programming language and stored in a source program storage section 1, for example, is input to a compiling apparatus 2 as is illustrated in FIG. 1. The compiling apparatus 2 performs word analysis, syntax analysis, semantic analysis and the like on the source program, thereby generating an executable program that can be executed in a target electronic computer. The executable program thus generated is stored into an executable program storage section 3.

The executable program stored in the executable program storage section 3 is executed, thereby acquiring data. The profile data of the executable program is

generated from the data thus acquired. The profile data is stored into a profile data storage section 4. The profile data contains various data items showing how the program worked when executed. For example, the data items show which part of the program has been executed, how many times it has been executed, and how long it has been executed each time.

As shown in FIG. 1, the source program written in a programming language and stored in the source program storage section 1 and the profile information stored in the profile data storage section 4 are input to the compiling apparatus 2. The compiling process is performed again, thereby optimizing the program. As a result, an optimal executable program is generated. The optimal executable program is stored into the executable program storage section 3.

To generate an optimal executable program, the source program is input to the compiling apparatus, thereby generating an executable program that can work in the target electronic computer, and the executable program is executed, thus acquiring data, and the profile data is generated from the data thus acquired, as has been s described above with reference to FIG. 1.

The source program and the profile data are input to the compiling apparatus and executed again. The compiling process is effected again, thereby generating an optimal executable program.

Thus, in order to acquire the profile data, the word analysis, syntax analysis, semantic analysis and the like must be performed on the source program to generate

the executable program, in the first compiling process. Further, these analyses must be performed again on the source program to generate the optimal executable program, in the second compiling process. Consequently, it would take much time to generate the optimal executable program.

Disclosure of Invention

In view of the above-mentioned problems with the prior art, the object of the present invention is to provide a compiling apparatus and a compiling method that can shorten the process time and can fast generate an executable program.

A compiling apparatus according to the present invention is characterized by comprising: an analysis data generating section for generating, form a source program, analysis data of the source program; a first executable program generating section for generating a first executable program on the basis of the analysis data; a profile data generating section for generating profile data on the basis of the first executable program; and a second executable program generating section for generating a second executable program on the basis of the analysis data and the profile data.

A compiling method according to the invention is characterized by comprising the steps of: generating, form a source program, analysis data of the source program; generating a first executable program on the basis of the analysis data; generating profile data on the basis of the first executable program; and generating section for generating a second executable program on the basis of the analysis data and the profile data.

Brief Description of Drawings

FIG. 1 is a diagram for explaining the sequence of generating an optimal executable program by the use of a conventional compiling apparatus;

FIG. 2 is a diagram for explaining a compiling apparatus according to one embodiment of the present invention;

FIG. 3 is a diagram for explaining the execution format generating section of the compiling apparatus shown in FIG. 2;

FIG. 4 is a diagram illustrating an example of a source program;

FIG. 5 is a diagram for explaining an example of an intermediate code, or analysis data generated by the source program analyzing section;

FIG. 6 is a diagram for explaining an example of an assembly code, or analysis data generated by the source program analyzing section;

FIG. 7 is a diagram for explaining an example of profile data generated by the profile data generating section;

FIG. 8 is a diagram for explaining an example of probability data generated by the probability data generating section of the execution format generating section;

FIG. 9 is a diagram for explaining an example of a program that has been optimized by using the probability data generated from the profile data;

FIG. 10 is a diagram for explaining another example of the execution format generating section incorporated in the compiling apparatus;

FIG. 11 is a diagram showing a compiling apparatus according to another

embodiment of this invention;

FIG. 12 is a diagram showing a compiling apparatus according to still another embodiment of the invention;

FIG. 13 is a diagram for explaining the designation data receiving section and execution format generating section of the compiling apparatus illustrated in FIG. 12;

FIG. 14 is a diagram for explaining an example of a graphical user interface, which is a program that assists compiling;

FIG. 15 is a diagram for explaining a program optimized in consideration of the compile instructing data given by an operator; and

FIG. 16 is a diagram for explaining a compiling apparatus according to another embodiment of the present invention.

Best Mode for Carrying out the Invention

Embodiments of the present invention will be described in detail, with reference to the accompanying drawings.

The compiling apparatus and the compiling method, both according to one embodiment of the invention, will be described, with reference to the drawings.

[First Embodiment]

FIG. 2 is a diagram explaining a compiling apparatus 1 according to the first embodiment of the invention. As is shown in FIG. 2, the compiling apparatus 100 is designed to generate an executable program that can be executed in a target electronic computer, from a source program stored in a source program storage section 10 and

written in a high-level programming language. The executable program thus generated is stored into an executable program storage section 20.

Moreover, the compiling apparatus 100 can generate profile data from the data that has been acquired by executing the executable program stored in the executable program storage section 20. In addition, the compiling apparatus 100 can generates an executable program by using the profile data, thereby to generate an optimal executable program.

The compiling apparatus 100, i.e., the first embodiment of the invention, will be described. As shown in FIG. 2, the compiling apparatus 100, or the first embodiment, comprises a profile data storage section 11, a source-program analysis data storage section 12, an execution format generating section 13, a profile data generating section 14, and a profile data storage section 15.

The source program stored in a source program storage section 10 and written in the high-level programming language is supplied to the source program analyzing section 11 of the compiling apparatus 100. The source program analyzing section 11 functions as an analysis data generating section. The section 11 performs various analyses, such as character analysis, syntax analysis, semantic analysis and the like, on the source program, thereby generating the analysis data of the source program. The source-program analysis data, thus generated, is stored into the source-program analysis data storage section 12.

The source-program analysis data stored in the source-program analysis data

storage section 12 contains the data generated by analyzing the source program, such as intermediate codes, assembly codes or machine words. That is, the analysis data contains data necessary to generate an executable program.

Then, the execution format generating section 13 generates an executable program that can be executed in the target electronic computer, from the analysis data stored in the source-program analysis data storage section. The executable program is stored into the executable program storage section 20. The executable program that can be executed in the target electronic computer is thereby acquired in the executable program storage section 20.

The executable program has been generated from the source program and has not been optimized. If the executable program is executed, the data processing will be wasteful, and high-speed data processing may not be achieved in some cases.

Therefore, the profile data generating section 14 of the compiling apparatus 100 generates profile data from the data that has been obtained by executing the target executable program stored in the executable program storage section 20. The profile data thus generated is stored into the profile data storage section 15.

In the compiling apparatus 100, the execution format generating section 13 uses the source-program analysis data acquired by the first compiling process and stored in the source-program analysis data storage section 12 and the profile data stored in the profile data storage section 15. Thus, the section 12 generates the executable program again.

The executable program is generated again in order to generate an optimal executable program. The process of generating the executable program again is carried out as illustrated in FIG. 2. First, the profile data stored in the profile data storage section 15 is used, thereby optimizing the program generated from the analysis data stored in the source-program analysis data storage section 12 and generating an executable program than can be executed at high speed without involving in wasteful data-processing. Then, this executable program is stored into the executable program storage section 20.

FIG. 3 is a diagram explaining the execution format generating section 13 of the compiling apparatus 100 according to the present embodiment. As is shown in FIG. 3, the execution format generating section 13 provided in this embodiment comprises a program forming section 131 and a program-execution probability data generating section 132. The program-execution probability data generating section 132 is designed to generate data representing the probability of executing a program, from the profile data. The program-execution probability data will be used to generate an optimal executable program.

As described above, the profile data of a target program has not generated yet in the profile data storage section 15 in the first compiling process. Therefore, in the first compiling process of the target source program, the execution format generating section 13 generates, for example, a program executable in the target electronic computer from only the analysis data stored in the source-program analysis data

storage section 12, such as the intermediate codes the assembly codes or the like. The executable program thus generated is stored into the executable program storage section 20.

Then, as indicated above, the profile data generating section 14 of the compiling apparatus 100 generates profile data from the data acquired by executing the optimal executable program stored in the executable program storage section 20. The profile data thus generated is stored into the profile data storage section 15.

The profile data stored in the profile data storage section 15 consists of the time data showing the time elapsed from the start of execution of the executable program and detailed data acquired at that time. As already mentioned, an optimal executable program is generated by performing again the process of generating an executable program, by using the profile data that is stored in the profile data storage section 15.

In this case, the analysis data stored in the source-program analysis data storage section 12, such as the intermediate codes the assembly codes or the like, is supplied to the program forming section 131. The program-execution probability data, which the program-execution probability data generating section 132 has generated by statistically processing the profile data stored in the profile data storage section 15, is supplied to the program forming section 131, too.

The program-execution probability data the program-execution probability data generating section 132 has generated is composed of various probability items. The probability items are, for example, the number of times the blocks constituting the

program have been executed, the number of times each command has been issued, the number of times branching is made, the number of accesses made to a memory and a register. These probability items will be described later.

The program forming section 131 of the execution format generating section 13 optimizes the program generated from the analysis data supplied from the source-program analysis data storage section 12, by using the probability data generated from the profile data stored in the profile data storage section 15. That is, the section 131 generates an executable program that can be executed at high speed, involving in wasteful data processing. The executable program is stored into the executable program storage section 20.

In this case, the source program analyzing section 11 carries out various analyses as described above, thus generating analysis data, which is stored into the source-program analysis data storage section 12. The analysis data is used, generating an optimal executable program. That is, the source program need not be repeatedly analyzed in order to generate an optimal executable program as in the conventional compiling apparatus. Hence, the time required to process data to generate an optimal executable program can be shortened. In other words, an optimal executable program can be generated fast.

The sequence of generating an optimal executable program in the compiling apparatus 100 according to the first embodiment comprises two processes, i.e., the first process and the second process, which will be described blow.

First, in the compiling apparatus 100, i.e., the first embodiment, the source program is read from the source program storage section 10 and subjected to various analyses, such as word analysis, syntax analysis, semantic analysis and the like in the source program analyzing section 11. The analysis data on the source program is thereby generated. The analysis data is stored into the source-program analysis data storage section 12 ((1) in FIG. 2).

Next, an executable program, which is designed for use in acquiring the profile data, is generated from the analysis data stored in the executable program is generated from the source-program analysis data storage section 12 ((2) in FIG. 2). The process performed until the executable program for acquiring the profile data, from the source program is the first process.

On the basis of the data acquired by executing this executable program generated in the first process, the profile data generating section 14 generates profile data ((3) in FIG. 2). The profile data is stored into the profile data storage section 15 ((4) in FIG. 2).

The execution format generating section 13 generates another executable program on the basis of the analysis data generated in the first process and stored in the source-program analysis data storage section 12 and also the profile data stored in the profile data storage section 15. That is, an optimal executable program is generated on the basis of the profile data. This executable program is stored into the executable program storage section 20 ((5) in FIG. 2). The process carried out after

the generation of the profile data, until the other executable program is generated, is the second process.

In the second process, it is unnecessary to analyze the source program since an optimal executable program is generated on the basis of the analysis data stored in the source-program analysis data storage section 12 and the profile data.

Thus, the source program need not be analyzed in the second process, if the source program has been analyzed in the first process. This shortens the time required for generate an optimal executable program. In other words, an optical executable program can be generated fast.

The compiling process effected in the compiling apparatus 100 according to the first embodiment will be explained in detail, with reference to an actual example.

FIG. 4 is a diagram illustrating the source program stored in the source program storage section 10. This example of a source program is designed to select one of variables a and b, depending on conditions, and to store the square of the variable selected, into a variable d.

As indicated above, the source program analyzing section 11 generates analyzes the source program stored in the source program storage section 10, thereby generating analysis data such as intermediate codes, assembly codes, machine words, or the like, and the analysis data (data generated) is stored into the source-program analysis data storage section 12. In this case, the user can select the kind of analysis data, intermediate codes, assembly codes, machine words or the like that should be

generated.

FIGS. 5 and 6 are diagram explaining examples of the analysis data stored in the source-program analysis data storage section 12, which has been generated by analyzing the source program stored in the source program storage section 10, by the source program analyzing section 11. Of these figures, FIG. 5 is a diagram showing the concept of an intermediate code, or analysis data. On the other hand, FIG. 6 is a diagram illustrating the concept of an assembly code, or analysis data.

In this instance, the BLK1of the intermediate code (FIG. 5) and the block 1 of the assembly code (FIG. 6) are each a block that compares variables a and b. The BLK2 of the intermediate code (FIG. 5) and the block 2 of the assembly code (FIG. 6) are each a block that contains if variable a is equal to or greater than variable b. The BLK3 of the intermediate code (FIG. 5) and the block 3 of the assembly code (FIG. 6) are each a block that stores variable b into variable c if the variable b is equal to or greater than variable a. The BLK4 of the intermediate code (FIG. 5) and the block 4 of the assembly code (FIG. 6) are each a block that stores the selected variable into variable d.

In FIG. 6, ld is a load command, st is a store command, cmp is a compare command, ble is a branch command, jmp is a jump command, and mul is a multiplication command. Hence, in the case of the assembly code used in the present embodiment, variable a is stored into a resister r1 in the first step of the block 1 and variable b is stored into a register r2 in the second step.

In the third step of the block 1, the values stored in the registers r1 and r2 are compared. The result of the comparison is stored into a register r3. Then, in the fourth step of the block 1, a jump is made to the block 3 if the result of comparison stored in the register is r1 ≤ r2 (variable a ≤ variable b). That is, the block 2 will be processed if r1 > r2 (variable a > variable b).

The block 2 is a process that will be carried out if the result of comparison between the values stored in the registers r1 and r2 is r1 > r2 (variable a > variable b) as has been described above. In the block 2, the variable a is stored into the variable a in the first step, and the value stored in the register r4 is stored into the variable c in the second step. Then, in the third step of the block 2, a jump is made to the block 4.

The block 3 is a process that will be performed if the result of comparison between the values stored in the registers r1 and r2 is r1 ≤ r2. In the block 3, the variable b is stored into a register r5 in the first step, and the value stored in the register r5 is stored into the variable c in the second step. Then, in the third step of the block 3, a jump is made to the block 4.

In the block 4, the variable c is stored into a register r6 in the first step, and the value c is stored into a register r7 in the second step. In the third step of the block 4, the value stored in the register r6 is multiplied by the value stored in the register r7. In the fourth step, the result of the multiplication effected in the third step is stored into a register r8.

As described above, the assembly code of the source program used in the

present embodiment, which illustrated in FIG. 6, is one designed to obtain the square of variable a or the square of variable b in accordance with the relation between variables a and b and to store the square of variable a or b into the register r8.

As indicated above, the execution format generating section 13 generates an executable program that can be executed in the target electronic computer, from the analysis data such as intermediate codes, assembly codes, machine words, or the like, which are stored in the source-program analysis data storage section 12. The executable program, thus generated, is stored into the executable program storage section 20. The executable program, now stored in the executable program storage section 20, is one that has been generated in the first compiling process; it is an executable program that has yet to be optimized.

Next, the profile data generating section 14 of the compiling apparatus 100 generates profile data from the data that has been acquired by executing the target executable program stored in the executable program storage section 20. The profile data is stored into the profile data storage section 15.

FIG. 7 is a diagram for explaining an example of the profile data generated by the profile data generating section 14 and stored in the profile data storage section 15. The profile data shown in FIG. 7 has been generated from the data that has been acquired by executable, for example, the executable program generated form the assembly code shown in FIG. 6.

As illustrated in FIG. 7, the profile data stored in the profile data storage section

15 is composed of the time that has passed since the start of executing the executable program and the detailed data acquired during that time. That is, the profile data shows how the executable program stored in the executable program storage section 20 has been executed. More precisely, the profile data indicates which commands have been executed, where the commands have branched, which memories have been accessed, and which registers have been accessed as time passed.

The example of profile data, shown in FIG. 7, is the detailed data showing the events that take place at various times since the start of executing the executable program, as will be explained below. That is, the example shown in FIG. 7 represents that the program counter (PC) changes at time 10050, whereby the block 1 of the assembly code shown in FIG. 6 is executed and variable a is loaded into the register. At time 10051, variable b is loaded into the register. At time 10053, a compare command (compare) is executed.

At time 10053, a branch command (branch less or equal) is executed, whereby the block 3 of the assembly code shown in FIG. 6 is executed. At time 10054, the program counter changes, and variable b is loaded into the register. At time 10055, the value in the register is stored into variable c. At time 10056, a jump command (jump) is executed in order to execute the block 4 of the assembly code shown in FIG. 6.

Thereafter, at time 10057, the program counter (PC) changes. The block 4 of the assembly code shown in FIG. 6 is thereby executed, loading variable c into the

register. At time 10058, variable c is loaded into the register.

At time 10059, a multiplication command (mul) is executed. At time 10060, the value of variable d, which has been obtained through the multiplication performed at time 10059, is stored into the register. Thus, the profile data, which is composed of the time that elapsed and the detailed data indicating how the executable program has been executed during that time, is stored into the profile data storage section 15.

In the compiling apparatus 100, the source program for the first compiling process is analyzed, thus generating analysis data. The analysis data is stored into the source-program analysis data storage section 12. The analysis data stored in the source-program analysis data storage section 12 and the profile data stored in the profile data storage section 15 are used, whereby the execution format generating section 13 generates an executable program again.

In this case, the analysis data, such as the intermediate code, assembly code or the like supplied from the source-program analysis data storage section 12, is supplied to the program forming section 131 of the execution format generating section 13. The probability data that has been generated by the program-execution probability data generating section 132 is also supplied to the program forming section 131.

FIG. 8 is a diagram for explaining an example of the probability data that the probability data generating section 132 has generated on the basis of the profile data stored in the profile data storage section 15. The example shown in FIG. 8 has been generated from the profile data (FIG. 7) obtained from the data that has been acquired

by converting the assembly code of FIG. 6 to an executable program and executing this executable program.

As is illustrated in FIG. 8, the probability data generating section 132 generates probability data by statistically processing the profile data stored in the profile data storage section 15 as described above. The probability data represents the number of times each block has been executed, the number of times each command has been issued, the number of times branching is made, the number of accesses made to the memory and the registers.

The probability data illustrated in FIG. 8 shows that the blocks 1 and 4 are executed every time the executable program is executed. The probability data also shows that probability that the block 2 is executed (that is, the probability that variable a is greater than variable b) is 10%, and that the probability that the block 3 is executed (that is, the probability that variable b is equal to or greater than variable a) is 90%.

The program forming section 131 of the execution format generating section 13 generates an optimal executable program on the basis of the analysis data stored in the source-program analysis data storage section 12. To be more specific, the section 131 generates an optimal executable program from the assembly code and the probability data shown in FIG. 8. The optimal executable program is stored into the executable program storage section 20.

FIG. 9 is a diagram for explaining an example of a program that has been optimized by using the probability data generated from the profile data. FIG. 9

illustrates, in the form of an assembly code, the optimal program generated by optimizing the assembly code shown in FIG. 6 by using the probability data, generated from the profile data.

As is shown in FIG. 8, the probability that the block 3 is executed is 90%, and the probability that the block 3 is executed is 10%. This means that the route of block 1 → block 3 → block 4, which has been optimized, is the optimal program shown in FIG. 9 (i.e., the assembly code).

As may be understood by comparing the optimal assembly code shown in FIG. 9 with the assembly code shown in FIG. 6, which is not optimized yet, the code for storing the value (i.e., variable b) in the register r2 into the register r5 is added at the third line of the assembly block 1 of the optimal assembly code. The code of the block 3 and the code of the block 4 will be integrated.

That is, when the program is optimized, the code of a part of the block 3 not optimized yet is moved to the block 1 already optimized. The code of the block 3 and that of block 4 are thereby integrated. At the same time, the allocation of registers is altered. The value (variable b) in the register r2 is thereby stored into the register r5. Therefore, the square of variable c (i.e., variable b) can be immediately obtained and stored into variable d in the block 3 that follows the block 1. In other words, the square of variable b can be obtained by processing two blocks, i.e., the block 1 and block 3.

If variable a is greater than variable b, variable b is stored into variable c via the

register r4 in the block 2, in the same way as in the assembly code of FIG. 6 that has yet to be optimized. In the block 4, variable a is multiplied by itself and the resultant square is stored into variable d. Thus, the square of variable a can be found by processing three blocks.

In this way, the executable program is optimized in consideration of the probability data generated from the profile data. On the basis of the fact that the probability that variable b is greater than variable a is as high as 90%, an optimal executable program is generated to perform data processing with high efficiency and at high speed in the case where variable b is greater than variable a.

Although not shown in FIG. 9, the block 5 to which the process jumps from the block 3 is one that follows the block 4. The execution format generating section 13 not only rearranges the commands, adds new commands, allocate the registers, and changes branch destinations, but also copies codes and deletes unnecessary codes. The section 13 can, therefore, generate an optimal executable program.

As described above, execution format generating section 13 incorporated in the compiling apparatus 100 according to this embodiment functions in the first compiling process as a section for generating an executable program that serves to generate profile data. After the profile data has been generated, the execution format generating section 13 uses the source program stored in the source-program analysis data storage section 12 and the probability data generated from the profile data stored in the profile data storage section 15, and functions as an optical program generating section for

generating an optimal executable program from.

The process of generating an executable program again is one designed to generate an optimal executable program. As illustrated in FIG. 2, this process is to optimize the program generated from the analysis data stored in the source-program analysis data storage section 12, by using the profile data stored in the profile data storage section 15. An executable program, which can be executed at high speed without involving in wasteful data processing, can thereby be generated and stored into the executable program storage section 20.

In this instance, the source program analyzing section 11 carries out various analyses, generating analysis data, and the analysis data is stored into the source-program analysis data storage section 12, as has been described above. The analysis data is used, thus generating an optimal executable program. In order to generate the optimal executable program it is not necessary to repeat the analysis of the source program as is required in the conventional compiling apparatus. The time that is spent to generate the optimal executable program can therefore be reduced. Thus, the optimal executable program can be generated fast.

There are two types of executable programs. One is a direct execution-format program, and the other is an indirect execution-format program. The execution format generating section 13 of the compiling apparatus 100 that is the first embodiment can generate both a direct execution-format program and an indirect execution-format program. It should be noted that a direct execution-format program can be executed

by some apparatuses, but cannot be executed by other apparatuses. On the other hand, an indirect execution-format program can be executed, not depending on the apparatus used.

Which type of a program, a direct execution-format program or an indirect execution-format program, should be generated can be automatically selected in accordance with, for example, the analysis data that is stored in the analysis data storage section 12. More specifically, a direct execution-format program will be generated if the analysis data stored in the analysis data storage section 12 is an assembly code or an object code designed to generate an executable program to be executed in the target electronic computer. If the analysis data stored in the analysis data storage section 12 is an intermediate code, an indirect execution-format program will be generated. The selection between the two types of executable programs can thus be achieved automatically.

Whether a direct execution-format program or an indirect execution-format program should be generated is selected by an alternative method. For example, a changeover switch provided on the compiling apparatus 100 may be operated to determine whether a direct execution-format program or an indirect execution-format program should be generated.

Similarly, an executable program of the first-process format, to be generated by analyzing the source program, or an executable program of the second-process format, to be generated by using the analysis data stored in the source-program analysis data

storage section 12, may be automatically selected when the compiling apparatus determines whether the program is the first one or any other one that will be compiled. Alternatively, whether an executable program of the first-process format or an executable program of the second-process format should be generated may be selected by operating the changeover switch provided on the compiling apparatus 100.

In the embodiment described above, the execution format generating section 13 can generate both an executive program for generating profile data and an optimal executable program. The present invention is not limited to that embodiment. Two execution format generating sections may be provided, one for generating profile data (i.e., a profile-data generating execution-format generating section), and the other for generating an optimal executable program.

As indicated above, the source program analyzing section 11 can generate intermediate codes, assembly codes, machine words, and the like. If the analysis data stored in the analysis data storage section 12 is, for example, an intermediate code, it will be necessary to convert the intermediate code to a code that may easily form an executable program and to convert the executable program, thus formed, to an object code. In such a case, the execution format generating section 13 needs to have a pre-code generator and a post-code generator.

FIG. 10 is a diagram for explaining another example of the pre-code generator and a post-code generator. This example of the section 13 designed to generate an executable program from the analysis data stored in the analysis data storage section

12 in the case where the analysis data is an intermediate code. In this case, a pre-code generator 133 is provided between the source-program analysis data storage section 12 and the program forming section 131, and a post-code generator 134 is provided at the output of the program forming section 131.

The pre-code generator 133 is designed to generate a code from the analysis data stored in the analysis data storage section 12. The code generated is of the format that can easily form an executable program in the program forming section 131. The post-code generator 134 is designed to generate an executable program of an object-executing format, from the executable program formed by the program forming section 131.

Therefore, the pre-code generator 133 converts the analysis data to, for example, an intermediate code from which an executable program can be easily generated, if the analysis data stored in the analysis data storage section 12 is, for example, a byte code of a specific programming language (e.g., Java). The intermediate code is supplied to the program forming section 131.

The program forming section 131 forms an executable program from the intermediate code supplied from the pre-code generator 131 by using, if necessary, the profile data. In this case, the executable program thus formed is not an object code. The post-code generator 134 therefore generates an executable program of an object-executing format, from the executable program that the program forming section 131 has formed from the intermediate code. The executable program the post-code

generator 134 has generated is stored into the executable program storage section 20.

Since the execution format generating section 13 has the pre-code generator 133 and the post-code generator 134, an executable program of an object-executing format can be generated even if the analysis data stored in the pre-code generator and a post-code generator 12 is an intermediate code or a Java byte code.

The pre-code generator 133 may be one that can generate an object code from the analysis data, such as an intermediate code, stored in the analysis data storage section 12. If so, the post-generator 134 need not be employed. Further, the program forming section 131 may be one that can form an executable program from, for example, a Java byte code or the like. If this is the case, it is unnecessary to use the pre-code generator 131, provided that the post-code generator 134 generates an executable program of an object-code executing format from the executable program generated by the program forming section 131.

In such a case, only the pre-code generator 133 may be provided, and to form an executable program from an intermediate code or the like, the intermediate code or the like is converted to an object code and the executable program is generated from the object code. Alternatively, only the post-code generator 134 may be provided, and to form an executable program from an intermediate code or the like, the executable program is generated directly from the intermediate code or the like and the post-code generator 134 converts the intermediate code or the like to an object code.

Still alternatively, there may be provided two execution format generating

sections, one having a pre-code generator and a post-code generator, and the other having neither a pre-code generator nor a post-code generator. In this case, an executable program can be generated from the analysis data stored in the pre-code generator and a post-code generator 12, no matter whether the analysis data is such an intermediate code as is shown in FIG. 4 or such an assembly code as is shown in FIG. 6.

The pre-code generator 133 and the post-code generator 134 are provided in the execution format generating section 13 in such a manner as shown in FIG. 10. Thus, an object code such as an assembly code, if supplied to the section 13, may bypass the pre-code generator 133 and also the post-code generator 134.

[Second Embodiment]

As has been described in conjunction with the first embodiment, the profile data is generated from the data acquired by executing an executable program. In some cases, however, the compiling apparatus cannot easily execute the executable program generated in it.

For example, the executable program generated may be one of the execution formats for a computer other than the apparatus that has performed the compiling process. Alternatively, the conditions in which the executable program will be executed, such as the command set or the OS (Operating System), are different because the hardware is different. In each case, the compiling apparatus cannot execute the executable program generated, and the profile data cannot be easily

acquired.

Further, the target executable program may be installed in the memory on the board of an electronic apparatus such as a VTR (i.e., the board of the target). In this case, it is necessary to install the executable program in a test board, to execute the program, and to supply the data acquired by executing the program to the compiling apparatus.

In this case, however, not only the test board must be prepared, but also the test board and the compiling apparatus must be connected by a number of cords. Moreover, a simulator needs to be mounted on the test board in order to execute the target executable program on the test board, thereby to acquire profile data. Much time and a great cost are required to develop the simulator.

Therefore, in the compiling apparatus according to the second embodiment, an intermediate code program of an indirect execution format is generated, which can be executed irrespective of the conditions in which the program will be executed. The compiling apparatus for generating an optimal executable program executes the intermediate code program, thereby acquiring profile data easily and quickly. The compiling apparatus can therefore generate the target executable program that is optimized.

FIG. 11 is a diagram showing a compiling apparatus 200 that can generate an intermediate code program of the indirect execution format. As is shown in FIG. 11, the compiling apparatus 200 comprises a source program analyzing section 11, a

source-program analysis data storage section 12, execution format generating sections 13A, 13B and 13C, a profile data generating section 14, a profile data storage section 15, and an indirect execution-format intermediate generating section 17.

The compiling apparatus 200 is identical to the compiling apparatus 100 according to the first embodiment, which has been described above, except for the indirect execution-format intermediate generating section 17 and the execution format generating sections 13A, 13B and 13C that generate executable programs from intermediate codes, assembly codes, machine words and the like. The components of the compiling apparatus 200 illustrated in FIG. 11, which are similar to those of the compiling apparatus 100, are denoted at the same reference numerals and will not be described in detail.

The execution format generating sections 13A, 13B and 13C are provided for generating programs of direct execution formats, respectively, which will be executed in different apparatuses. Although the execution format generating sections 13A, 13B and 13C are designed to generate executable programs of different formats, they can use the same profile data.

In the present embodiment, the execution format generating sections 13A uses an intermediate code, the execution format generating sections 13B uses an assembly code, and the execution format generating sections 13C uses another object code. That is, the sections 13A, 13B and 13C use different analysis data items, in accordance with the executable programs they will generate.

Nonetheless, the execution format generating sections 13A, 13B and 13C have the same basic structure as the execution format generating section 13 of the first embodiment, which has been described with reference to FIG. 3 and FIG. 10. Thus, the execution format generating sections 13A, 13B and 13C can generate optimal executable programs by using the probability data generated from the profile data stored in the profile data storage section 15.

In the compiling apparatus 200 shown in FIG. 11, the indirect execution-format intermediate generating section 17 generates an intermediate code program that can be executed in the compiling apparatus 200, from the source-program analysis data stored in the source-program analysis data storage section 12. The intermediate code program thus generated is stored into an indirect execution-format intermediate program storage section 30.

The intermediate code program stored in the indirect execution-format intermediate program storage section 30 is executed, generating data. From this data, the profile data generating section 14 generates profile data. The profile data is stored into the profile data storage section 15.

Any one of the execution format generating sections 13A, 13B and 13C generates an optimal executable program of indirect execution format from the profile data stored in the profile data storage section 15 and the analysis data stored in the source-program analysis data storage section 12. The optimal executable program, thus generated, can be executed in the target computer. In the second embodiment,

one of the execution format generating sections 13A, 13B and 13C is selected to be used, by, for example, the operator of the compiling apparatus 200.

Even if the executable program, which the compiling apparatus has generated in order to generate the profile data for optimization, cannot be executed at once in the compiling apparatus, it is possible to generate an intermediate code program of the indirect execution format that can be executed by any computer. When the intermediate code program is executed, the profile data for the target program is acquired. An executable program of the indirect execution format, which can be executed in the target computer, can be therefore generated fast.

In the second embodiment, the executable programs generated by the execution format generating sections 13A, 13B and 13C, respectively, are stored into executable program storage sections 20A, 20B and 20C, respectively. The executable program the operator wants can thereby be generated.

In the embodiment described above, the operator selects one of the execution formats generating sections 13A, 13B and 13C, which will be used. The present invention is not limited to this, nonetheless. For example, the operator may select analysis data to be used, such as an intermediate code, an assembly code, a machine word or the like, automatically selecting one of the execution format generating sections. In other words, the compiling apparatus 200 can be designed to use analysis data items and the execution format generating sections in one-to-one correspondence.

Alternatively, the compiling apparatus 200 may be designed, enabling the

operator to select both an analysis data item and an execution format generating section, so that the target executable program may be generated.

Still alternatively, a multi-function execution format generating section may be provided in place of a plurality of execution format generating sections. In this case, the multi-function execution format generating section generates an executable program from the analysis data supplied to the section in accordance with the instruction made by the operator, no matter whether the analysis data is an intermediate code, an assembly code, a machine word, or any other object code.

Thus, the compiling apparatus 200 according to the second embodiment can generate the target executable program that the operator wants, in accordance with the instruction made by, for example, the operator.

The analysis data supplied to any execution format generating section may be an intermediate code, an assembly code, a machine word or the like, as the operator desires. In a similar way, an intermediate code, an assembly code, a machine word or the like may be selected and supplied to the indirect execution-format intermediate generating section 17.

Therefore, the analysis data supplied to the indirect execution-format intermediate generating section 17 and the analysis data supplied to the execution format generating sections 13A, 13B and 13C are codes of different types in some cases. In other cases, the analysis data supplied to the section 17 and the analysis data supplied to the sections 13A, 13B and 13C are codes of the same type. In short, since

various analysis data items are selectively used, programs of various indirect execution formats (i.e., intermediate code programs) and programs of various direct execution formats can be generated more easily.

Whether the analysis data supplied to the section 17 should generate an intermediate code program or any one of the execution format generating sections 13A, 13B and 13C should generate an executable program may be selected by operating, for example, the changeover switch that is provided on the compiling apparatus 200.

[Third Embodiment]

One compiling apparatus may be designed to generate executable programs of direct execution format, executable programs of indirect execution format, and also to generate executable programs that will be executed in various types of computers in different conditions. If this is the case, the data designating the direct execution format or indirect execution format and to the data representing the conditions in which the programs will be executed in the target computer compiling apparatus need to be input to the compiling apparatus.

Therefore, the compiling apparatus according to the third embodiment is one that receives the compile designation data generated by the operator of the compiling apparatus or the compile designation data generated from the program that assists the compiling process. The compiling apparatus can generate an executable program in accordance of the compile designation data, too.

FIG. 12 is a diagram showing the compiling apparatus 300 according to the embodiment of the present invention. As illustrated in FIG. 12, the compiling apparatus 300 comprises a source program analyzing section 11, a source-program analysis data storage section 12, an execution format generating section 13, a profile data generating section 14, a profile data storage section 15, and a designation data receiving section 18.

All these sections, except the designation data receiving section 18, are identical in structure to their counterparts provided in the compiling apparatus 100 according to the first embodiment. Therefore, the components of the compiling apparatus 300, i.e., the third embodiment illustrated in FIG. 12, which are similar to those of the compiling apparatus 100, i.e., the first embodiment shown in FIG. 2, are designated at the same reference numerals and will not be described in detail.

This embodiment will be described, assuming that the source program storage section 10 stores the source program shown in FIG. 4, that the source program analyzing section 11 analyzes the source program, generating, for example, an assembly code of the type shown in FIG. 6, and that the analysis data storage section 12 stores the assembly code.

The operator 31 operates a pointing device such as a keyboard device or a mouse is operated, generating compile designation data. Alternatively, compile designation data may be supplied by the use of a compile-assisting program such as a graphical user interface (GUI) 32. The designation data receiving section 18

receives the compile designation data and supplies the same to the execution format generating section 13.

The compile designation data is one that designates the indirect execution format or the direct execution format or one that controls the compiling process. The data for controlling the compiling process is, for example, data that controls arrangement of codes, data that controls the optimization of a specific block, or the like.

The designation data receiving section 18 also receives compile designation data generated from the program that assists the compiling process. This program provides the data designating the file that contains the parameters required in the compiling process to generate the target executable program and the data required in the compiling process. The designation data receiving section 18 supplies the compile designation data to the execution format generating section 13.

On the basis of the compile designation data supplied from the designation data receiving section 18, too, the execution format generating section 13 generates an executable program from the analysis data stored in the source-program analysis data storage section 12 or from both the analysis data stored in the source-program analysis data storage section 12 and the profile data stored in the profile data storage section 15.

An optimized executable program, which is the target program, can thereby generated fast and easily on the basis of the compile designation data given by the

operator or generated from the program that assists the compiling process.

The compiling apparatus 300 according to the third embodiment will be described in detail, with reference to an actual example.

To the compiling apparatus 300, i.e., the third embodiment of the invention, a display is connected. The display displays the various data items. The data items are, for example, the analysis data acquired in the source program analyzing section 1, the executable program generated in the execution format generating section 13, and the probability data generated by statistically processing the profile data. The operator can therefore know these data items.

Seeing and confirming the data items displayed, the operator can input various compile designation data items. That is, the graphical user interface 32 functions in the third embodiment. It is through the graphical user interface that the compile designation data is input to the compiling apparatus 300.

In accordance with, for example, the data displayed by the display, the operator operates the pointing device such as a keyboard device or a mouse, generating compile designation data. Alternatively, the operator inputs the compile designation through the graphical user interface. The compile designation data thus input is supplied via the receiving section 18 to the execution format generating section 13.

FIG. 13 is a diagram for explaining the designation data receiving section 18 and execution format generating section 13 of the compiling apparatus 300 according to the third embodiment. The execution format generating section 13 receives the

compile designation data from the designation data receiving section 18. The compile designation data may be one designating either the indirect execution format or the direct execution format. If so, the compile designation data received will be supplied to the program forming section 131 of the execution format generating section 13.

The compile designation data received may be one that changes the probability data generated from the profile data to control the optimization of an executable program and to generate an optimal executable program. In this case, the data received is supplied to the probability data generating section 132.

FIG. 14 is a diagram showing an example of the graphical user interface 32 displayed. Namely, FIG. 14 illustrates the screen displaying the branch probability which is a part of the probability data (FIG. 8) the probability data generating section 132 has generated from the profile data. In accordance with the data displayed, the branch probability will be changed.

As indicated above, the probability that the block 2 is executed (variable a is greater than variable b) is 10% in the case of the assembly code shown in FIG. 9. The probability is displayed as T = 10% as is illustrated in FIG. 14. On the other hand, the probability that the block 3 is executed (variable b is greater than variable a) is 90% in the case of the assembly code shown in FIG. 9. The probability is displayed as F = 90% as is illustrated in FIG. 14.

To perform optimization, the branch probability is changed in the data change input region for controlling the optimization, as is indicated by the arrow in FIG. 14.

In the case of in FIG. 14, the probability that the block 2 is executed (variable a is greater than variable b) is changed to 90%, and the probability that the block 3 is executed (variable b is greater than variable a) is changed to 10%.

The data for controlling the optimization is supplied to the probability data generating section 132 of the execution format generating section 13, as has been explained with reference to FIG. 13. The branch probability data is thereby changed. As has been described, the execution format generating section 13 generates an executable program in accordance with the program for assisting the compiling process, which has been supplied from the designation data receiving section 18. The program for assisting the compiling process is, for example, the compile designation data supplied from the graphical user interface.

FIG. 15 shows an executable program optimized by changing the branch probability as has been explained with reference to FIG. 14. The executable program thus optimized is an assembly code. The assembly code not optimized yet is shown in FIG. 6. The route of block 1 → block 2 → block 3 is optimized as is illustrated in FIG. 15.

As can be seen by comparing the assembly code shown in FIG. 15 with the assembly code shown in FIG. 6 and not optimized yet, a code for storing the value (variable a) in the register r1 into the register r4 is added to the third line of the block 1 of the assembly code when the assembly code is optimized by using the probability data. The code of the block 2 and the code of the block 4 are thereby integrated.

The value (variable a) in the register r1 is now stored in the register 4 in the block 1. Therefore, the square of variable a can be obtained at once and stored into variable d in the block 2 that follows the block 1, if variable a is greater than variable b. That is, the square of variable 2 can be obtained by processing two blocks.

Variable b may be greater than variable a. If so, variable b is stored into variable c in the block 3 and the square of variable c (i.e., variable) is obtained and stored into variable d, as in the assembly code not optimized yet. Thus, the square of variable a can be obtained by processing three blocks.

Hence, an optimal executable program is generated so that the process may be performed fast when variable a is greater than variable b, which may occur at the probability of 90%. Although not shown in FIG. 15, the block 5 to which the process jumps from the block 2 is one that follows the block 4.

As described above, the compiling apparatus 300 according to the third embodiment can generate an optimal executable program, i.e., the target program, easily and fast, in accordance with the compile designation data, too, which has been supplied from the program for assisting the compiling process. That is, the operator performs manual operation, giving the compile designation data to the compiling apparatus 300. Minute controls, which the compiling apparatus 300 cannot automatically perform, can therefore be effected in the apparatus 200. The compiling apparatus 300 can generates an executable program with higher efficiency than otherwise.

Furthermore, the operator can supply an instruction to the execution format generating section 13 through the designation data receiving section 18. The instruction designates that an executable program should be generated on the basis of the analysis data stored in the analysis data storage section 12 in order to acquire the profile data. Alternatively, the instruction designates that an optimal executable program should be generated on the basis of the profile data.

[Fourth Embodiment]

In the third embodiment described above, the compile designation data is supplied to the execution format generating section 13 through the designation data receiving section 18 in order to perform the compiling process. In most cases, however, the compile designation data is unknown before the compiling process.

It is therefore desired that any compile designation data that is known before the compiling process be supplied to the compiling apparatus. If such compile designation data is supplied to the apparatus, the compiling apparatus can effect the compiling process quickly. If the compile designation data happens to have an error, it may be corrected, whereby the apparatus can perform the compiling process without inputting the entire compile designation data again. Hence, the compiling apparatus can generate an executable program fast.

Therefore, the compiling apparatus according to the fourth embodiment has a compile designation data storage section for holding the compile designation data that assists the operator or facilitates the compiling process. Thus, the compile designation

data need not be input every time the compiling process is carried out.

FIG. 16 is a diagram for explaining the compiling apparatus 400 according to the fourth embodiment. As shown in FIG. 16, the compiling apparatus 400 according to the fourth embodiment is similar in structure to the compiling apparatus 300, i.e., the third embodiment described with reference to FIG. 12. The compiling apparatus 400 is identical to the compiling apparatus 100 according to the first embodiment, except for the use of an designation data receiving section 18 and a compile designation data storage section 19.

In the compiling apparatus 400 according to the fourth embodiment, the designation data receiving section 18 receives the compile designation data generated from the program that assists the compiling process. The compile designation data is stored into the compile designation data storage section 19.

The compile designation data stored in the compile designation data storage section 19 is read out by the execution format generating section 13 in order to generate an executable program. The execution format generating section 13 generates an executable program in accordance with the compile designation data read from the compile designation data storage section 19, too.

Since the compile designation data generated from the program that assists the compiling process is stored in the compile designation data storage section 19, it is not necessary to input many compile designation data items to perform the compiling process. Nor is it necessary to execute the program that assists the compiling process.

The executable program, i.e., the target program that can be executed in the target computer, can therefore be generated faster.

That is, the compile designation data to be supplied to the compiling apparatus to start the compiling process can be minimized. The time required for inputting the compile designation data can therefore be shortened. As a result, the target executable program can be generated fast.

The source program, the intermediate code, the assembly code, the profile data, the probability data and the like, all used in the embodiments described above, are nothing more than examples. The compiling apparatus according to this invention can generate analysis data, executable programs, profile data, and optimal executable programs, from various source programs.

The compile designation data is not limited to data that designates an execution format, data representing the conditions of executing a program in the target computer, and data for controlling the compiling process. Rather, the compile designation data includes various data items that are necessary in the compiling process of generating the target executable program.

The analysis data supplied to the indirect execution-format intermediate generating section 17 used in the second embodiment described above may be provided in the compiling apparatuses 300 and 400 which are the third embodiment and the fourth embodiment, respectively. If this is the case, the instruction the operator has input to generate an intermediate code program is supplied to the analysis

data supplied to the indirect execution-format intermediate generating section 17, which generates an intermediate code program.

The compiling apparatuses according to the embodiments described above can be constituted of software. In other words, a compiling program can be generated that has the functions of the compiling apparatuses according to the embodiments described above.

The execution format generating sections incorporated in the compiling apparatuses according to the embodiments described above can link a plurality of executable programs associated with one another, thereby to constitute one module of a specific execution format.

The compiling apparatus according to each embodiment described above can operate by itself. Nonetheless, it may be incorporated into various electronic computers and be used therein.

Further, the compiling apparatus according to each embodiment described above can generate an executable program that can be executed in the target computer, regardless of the data configuration of the program the target computer uses.

That is, the compiling apparatus according to each embodiment can generate an executable program that can be executed in an electronic apparatus such as the target computer, not influenced by the hardware configuration of the computer incorporating it or the data configuration of the program used in the target computer.

As has been described above, in the compiling apparatus according to the

present invention, analysis data is generated from a source program, a first executable program is generated on the basis of the analysis data, profile data is generated on the basis of the first executable program, and a second executable program is generated on the basis of the analysis data and profile data. Hence, the source program need not be used repeatedly in order to generate an optimal executable program. That is, an optimal executable program can be generated on the basis of the source-program analysis data generated by analyzing the source program and the profile data. To generate an optimal executable program, the source program need not be analyzed again after it has been analyzed, thereby generating the first executable program from which the profile data has been generated. The optimal executable program can therefore be generate fast.

In the compiling apparatus according to the present invention, the above-mentioned analysis data is stored, too. Therefore, one and the same execution format generating section can generate not only the executable program which is not optimized and from which the profile data is generated, but also the executable program, which has been optimized. Two execution format generating sections need not be provided, one for generating the executable program from which the profile data is generated, and the other for generating the executable program, which has been optimized.

The compiling apparatus according to the present invention can generate an executable program which has the format specific to the target computer and which

complies with the conditions in which the target computer executes programs, thanks to the above-mentioned profile data stored in a memory. When this program of direct execution format is executed in an apparatus having a compiling apparatus, the profile data of the direct execution-format program is acquired. An optimal executable program can thereby generated.

Moreover, the compiling apparatus according to the present invention can generate a program of an indirect execution format, too. The program of the indirect execution format can be executed, regardless of the type of the computer that executes it. The compiling apparatus executes the indirect execution-format program, acquiring profile data. The profile data is used, too, whereby an optimal executable program can be generated.

In the compiling apparatus according to the present invention, the so-called intermediate code program of an indirect execution format, which has been generated by the indirect execution-format intermediate program storage section, can be executed irrespective of the conditions in which the program will be executed. The profile data of the program is thereby acquired. The profile data is used, whereby an optimal executable program, i.e., target program, is generated both easily and quickly.

The compiling apparatus according to this invention can generate an executable program easily and fast, in accordance with the compile designation data supplied from the operator, too.

With the compiling apparatus according to this invention, the operator can store

prescribed compile designation data into the compile designation data storage section prior to the compiling process. This shortens the time required to input the compile designation data when the compiling process is to be started. Thus, an executable program that complies with the operator demand can be generated easily and fast in accordance with the compile designation data stored in the compile designation data storage section.

Further, the compiling apparatus according this invention generates an executable program on the basis of the compile designation data generated from the program that assists the compiling process. The apparatus can therefore generate an optimal executable program, i.e., target program, easily and quickly.

Still further, in the compiling apparatus according to the present invention, the compile designation data storage section can store various items of compile designation data, which have been generated from the program that assists the compiling process. It is therefore unnecessary to execute the program for assisting the compiling process, in the course of the compiling process. This makes it possible to generate a target executable program faster, in accordance with the compile designation data stored in the compile designation data storage section.